



UNIVERSITY OF
ILLINOIS LIBRARY
AT URBANA-CHAMPAIGN
ENGINEERING

510.84
I 263e
no. 234
C 1

ENGINEERING LIBRARY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS

Enjin

CONFERENCE ROOM



Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document Number 234

**Optimization of a Relational-Algebra
Query to a Distributed Database
Using Statistical Sampling Methods**

by

David A. Willcox

August 1, 1977

The Library of the

MAY 23 1978

University of Illinois
at Urbana-Champaign

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

ENGINEERING

JUN 6 1978

Optimization of a Relational-Algebra Query
to a Distributed Database
Using Statistical Sampling Methods

by

David A. Willcox

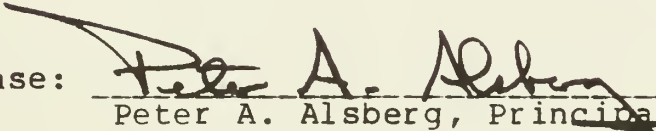
Prepared for the
Command and Control Technical Center
WWMCCS ADP Directorate
of the Defense Communications Agency
Washington, D.C.

under contract
DCA100-75-C-0021


Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

August 1, 1977

Approved for release:



Peter A. Alsberg, Principal Investigator



Digitized by the Internet Archive
in 2012 with funding from
University of Illinois Urbana-Champaign

<http://archive.org/details/optimizationofre00will>

Table of Contents

	Page
I. INTRODUCTION	1
II. STATEMENT OF THE PROBLEM	5
Representation of the Query	5
Distributed Database	6
A Simplifying Assumption	7
Distributed Optimization	8
III. STATE OF THE ART OF RELATIONAL QUERY OPTIMIZATION	10
Single-Site	10
Multiple-Site	13
IV. SAMPLING	16
Sampling for Simple Select Functions	17
Join Prediction	26
Generalized Query Sampling	31
Prediction of Project Output	40
Pathological Cases	41
V. AN INTEGER LINEAR PROGRAM FOR OPTIMAL QUERY STRATEGIES	44
Definition of the Integer Linear Program	44
Notation	47
Solving the ILP as a Linear Program	48
An Algorithm for Finding Integer Solutions	58
VI. CONCLUSIONS	60
REFERENCES	61
APPENDIX A	62

I. INTRODUCTION

In this thesis I will consider the problem of optimizing a query expressed in a particular canonical form when the data needed to service the query is distributed among several distinct computers. The optimization is intended to minimize (or at least reduce) the total cost of servicing the query. The individual "host" computers can transfer data among themselves via some communication facility such as phone lines or a packet-switched network. On the assumption that communication between computers is relatively slow and costly (in ARPANET, for instance, host-host bandwidth is at least 50-100 times slower than that between a computer and its local disk system), the global optimization problem is primarily that of minimizing the amount of data which must ultimately be transferred between computers. All other factors will have a relatively minor effect on query cost. I am concerned primarily with this global problem of reducing network traffic resulting from the given query. Local optimization, or optimization of that part of the query which involves only a single site, is not considered. A fairly intuitive description of the problem and my approach to its solution are given below.

Consider the following problem. A company maintains a database with information on parts supplied by a group of suppliers. The database is split into two pieces. One piece, called PARTS, contains descriptions of all of the parts used by the company in its operations. The other piece, called SUPPLIERS, contains information on each supplier together with a list of parts it supplies. (For each supplier, there is one

record for each part it supplies.) It is necessary to get a list of all suppliers in Illinois which supply square, green parts.

If both pieces are stored on the same computer, this query can be serviced in a straightforward manner by

- 1) Looking thru PARTS and constructing SG-PARTS, a list of all square, green parts,
- 2) Looking thru SUPPLIERS and constructing I-SUPPLIERS, a list of Illinois suppliers and the parts they supply, and
- 3) Combining the two lists to find suppliers in I-SUPPLIERS who supply parts listed in SG-PARTS.

Suppose, however, that the company owns two computers. These computers can communicate with each other via a medium speed communication line. Further suppose that each computer has only one of the two pieces of the database. How then can the query best be serviced? It would be possible, of course, to send one of the two pieces over the line and then process the query as before. However, if the pieces are very large, this would be very expensive.

A much better approach would be to form the two lists SG-PARTS and I-SUPPLIERS on the computers with the respective pieces and then send one of the lists. On the assumption that these restricted lists are much smaller than the original database pieces (they certainly can't be any larger), this will greatly reduce the time and cost to service the query. Since either SG-PARTS or I-SUPPLIERS must be transmitted, but not both, it is clear that the cheapest way to service the query would be to ship the smaller of the two lists.

It would be possible, especially in this simple case, to decide which list is smaller by first forming the lists on their respective hosts and then comparing their sizes. However, this technique does not easily generalize to the more complex queries which will be considered later. In this thesis I will describe a more general approach which will allow the decision to be made before any significant processing of the actual query is started. This will necessitate a priori estimates of the sizes of SG-PARTS and I-SUPPLIERS. The "classical" method for making this prediction involves the assumption of independence between fields of a database. (See, for example, [Wong and Youssefi 76], [Hammer and Chan 76], and [Vallarino 76].) If it is known that 25% of all parts are green and 20% of parts are square (ignoring the question of where that information is obtained), the independence assumption implies that $25\% \times 20\% = 5\%$ of all parts are square and green. This assumption has one major drawback. There is no way to get any quantitative estimate or bound for the error in the estimate. There will be cases where independence is a poor model of the state of things, and there is no way to know, without actually running the query, when a given query has hit such a case.

Instead of assuming independence, it is possible to forecast the query performance by taking a small random sample of the database pieces and using those samples to estimate the sizes of the intermediate results. To estimate the size of SG-PARTS, for instance, one could take a small sample of PARTS and find out what percent of the parts in the sample are square and green. If there are 10,000 parts, and 10% of the sampled parts are square

and green, then it is reasonable to assume that about 10% (or a total of 1000) of all parts are square and green. This estimate won't always be accurate, either. However, it is possible to obtain a precision for the estimate using well-known statistical techniques. This precision will depend on several factors, including the size of the sample and the estimate itself. It is expressed as an absolute error bound and a "level of confidence" that the actual error is less than the given error bound.

In this thesis, I extend this approach to a general query expressed in Codd's relational algebra [Codd 70]. By taking appropriate samples of the pieces in the database, it is possible to estimate the volume of output from each intermediate operation in the query. These volume estimates can be used to develop the parameters to an integer linear programming (ILP) problem which can be used to find an optimal strategy for the query in its specified form. The resulting strategy might not be globally optimal because permutations and other manipulations of the query operations are not considered, but it will usually find the best strategy based upon the given order of operations.

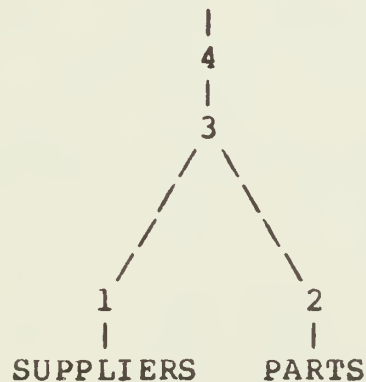
Because setting up the database samples will be fairly expensive, and because solving the ILP is expensive (even with the simplification which I will present), the optimization technique will be useful only for fairly large, expensive queries. When the database is large, the expense incurred in doing the optimization can be recovered by improved query performance.

II. STATEMENT OF THE PROBLEM

Representation of the Query

The queries discussed in this thesis will be expressed in the relational algebra first described by Codd [70]. Any reader not familiar with the relational database model should refer to that paper first. The important concepts for understanding this thesis are those of a relation, tuple and domain, and the relational operators join, select (also called restrict) and project. The exact format in which the query is expressed by the user is unimportant. It is assumed here that before any optimization is attempted the query will be parsed into an internal tree format. Each internal tree node represents a relational operator, and each terminal node is a relation from the database. The output from any relational operator is a relation, so each intermediate operand in the query (the output of any operator) is a relation.

The query in the supplier-parts example of Chapter I might be represented as a tree something like this:



The operations at the internal, numbered nodes are as follows:

Node 1 - Select tuples with STATE="Illinois".

Node 2 - Select tuples with COLOR="green" and SHAPE="square".

Node 3 - Join on domain SUPPLIER#.

Node 4 - Project out domain NAME.

Distributed Database

The relations which make up the database in question are assumed to be distributed among a number of different computers. Some kind of communication facility (the "network") is assumed to exist. This could be either a packet-switched, store-and-forward network (e.g. ARPANET or CYCLADES), or a set of dedicated phone lines. The network allows each computer (called a "host") to communicate with any other computer on the network. Therefore, all of the database relations are available to every computer on the network, even though no one computer has copies of all of the relations.

It is also possible that several copies of each relation are stored at different computers. A database system which allows multiple copies of the relations has two advantages over one that maintains only single copies. The major advantage is that this will increase the availability of the data to users. If there are several copies of each relation, the probability that data is unavailable due to a computer (or computers) being down is greatly reduced. If each computer is up 90% of the time, then each relation in a single-copy system would be available 90% of the time. If two copies of each relation were maintained, then each relation would be available 99% of the time, assuming

that the availabilities of the different machines are independent.

A second advantage of a multicopy system is that the possibilities for reducing query costs are increased. If there are several copies of each file, it is more likely that there will be a copy where it is most needed to allow a cheap strategy for a given query. This is a purely intuitive argument, and no attempt has been made to evaluate how much effect multiple copies will have on the optimal strategy. In no event will multiple copies degrade the performance of a retrieval, however.

A Simplifying Assumption

The database considered here is one which is derived from some original "goal" relation GOAL. Each database relation represents a small subset of GOAL obtained by picking a subset of the domains and then removing duplicate tuples. Splitting the large goal relation up into subrelations has two advantages. If the goal relation is not in Codd's third normal form, then it is possible to split it up in such a way as to produce subrelations which are in third normal form (assuming that the functional dependencies between domains are known) [Codd 72]. This process is called normalization. It is much easier to maintain the internal consistency of a database which has relations in third normal form than one which does not. Also, if GOAL is unnormalized, this implies that there is at least some redundancy in the data. The data which represent some relationship may be replicated many times. Decomposing GOAL into normalized

subrelations will remove these redundancies.

It is possible to split up any goal relation into smaller relations in such a way as to preserve the functional relationships between domains. (Bernstein [76] gives one algorithm to do this normalization.) These database relations collectively will contain all of the information contained in the goal relation, so it would be possible to reconstruct the goal relation exactly by recombining the database relations. Unfortunately, there is no guarantee in the general case that combining two of the database relations using a "natural" join will produce a result which accurately reflects the original data. It might be possible to get invalid associations between domain values.

The query operations I consider will be limited to those which reconstruct a portion of GOAL. This places no restriction on the possible select and project operations, but it does constrain the choice of joins. It implies that the join of any pair of relations is unique. There could be several ways to obtain this join (it might, for instance, be possible to use any of several equivalent domains as the joining domain) but the result will be the same however it is produced. The join of any group of domains is also unique. This implies that, for the set of allowed operations, the joins are associative.

Distributed Optimization

The bandwidth for communications between computers in the network is very low compared to the bandwidth of local disk

accesses. Data rates of tens of kilobaud are typical for networks as compared to a few megabaud for typical disks. On the assumption that processing within the CPU takes place at a much higher rate than input/output, it follows that a minimization of query response time can be reduced to a minimization of total network traffic. Wong [77] makes a similar argument that minimization of network traffic will minimize query cost. Therefore, purely local considerations will have a minimal effect upon the allocation of query operations to network hosts and can be ignored. (Local optimization can, of course, be performed after global optimization to improve the performance of that part of a query which ends up at a given host.) To minimize total network traffic, it is clearly necessary to be able to estimate the sizes of the intermediate operands (the objects which might get transmitted) in the query. Chapter IV describes a method for making such estimates. In that chapter, a procedure is described which will produce samples of the individual database relations. By appropriately interpreting the results of a query run against this database sample, it will be possible to forecast the result of running the query against the whole database. Chapter V describes an integer linear programming model which will yield an optimal query strategy based upon the results of sampling. But first, some of the approaches to distributed and/or relational query optimization which have appeared previously in the literature are discussed in Chapter III.

III. STATE OF THE ART OF RELATIONAL QUERY OPTIMIZATION

Single-Site

Most of the work on optimization of relational queries has been concerned with single-site databases only. As is pointed out by Wong [77], the problems involved in optimizing a distributed query are much different from those encountered in the single-site case. Therefore, the two results described here are given primarily for background purposes.

Tree Permutations. One form of optimization, described by Smith and Chang [75], involves the rearrangement of the order in which operations are performed so the cost of the query is reduced. The procedure they describe works on a query expressed as a tree. Each node of the tree is an operator, such as project or join. They identify a number of possible types of tree permutations which have the effect of altering the order in which the operations are performed. They also identify a number of different algorithms for performing each type of operation. Associated with each algorithm is a measure of its relative cost and any requirements on each of its input relations. (The cheapest algorithm for join, for instance, can only be used if one of the input relations has an index available for the joining domain.) Also, relevant attributes of the output from each algorithm are noted. For example, the output might be sorted on some domain value. The general effect of their procedure is to perform inexpensive operations, such as restrict and project, as early as possible to reduce the total number of tuples in

consideration, without masking useful features like indexes from the more expensive operations.

A simplified version of this procedure could be used as a first step in optimizing a distributed query. Such factors as the specific algorithm used for performing each operation and the presence or absence of indexes can probably be ignored, but it will be useful to "move down" the project and restrict operations as low in the tree as possible.

Query Decomposition. Another approach, described by Wong and Youssefi [76], works on the very different kind of query used in INGRESS. The user's view of the database in INGRESS is simply the cartesian product of all of the relations in the database. The query is expressed in the form of a restrict on this cartesian product. There is no join operation per se. An equivalent function is performed by an appropriate select function on the joining domain. The query includes a list of domains to be "projected out" from the user view after the select is applied. The example from Chapter I (a list of Illinois suppliers who supply square, green parts) would be expressed in INGRESS in a form something like this:

```
RETRIEVE (SUPPLIERS.NAME) WHERE
    (SUPPLIERS.STATE = "Illinois"
    & PARTS.COLOR = "green"
    & PARTS.SHAPE = "square"
    & PARTS.SUPPLIER# = SUPPLIERS.SUPPLIER#)
```

A query in this form is decomposed by alternating between "detachment" and "tuple substitution". Detachment is the process of splitting off subqueries which have only one relation in

common with the rest of the query. (That is, there is only one relation which appears in both the detached subquery and the part of the original query that is left after detachment.) Tuple substitution is the successive substitution of tuple values retrieved by one subquery into the expression for another subquery. In effect, this produces a separate subquery for each tuple value substituted. Generally, there will be many different possible decompositions of any given query. One possible decomposition of the above sample query will yield two successive subqueries, the first of which is

```
RETRIEVE (PARTS.SUPPLIER#) INTO(temp) WHERE
  (PARTS.COLOR = "green"
   & PARTS.SHAPE = "square")
```

This query produces a list in "temp" of supplier numbers for suppliers who supply square, green parts. For each supplier number "supplier#" in temp, tuple substitution would generate a query in the form

```
RETRIEVE (SUPPLIERS.NAME) WHERE
  (SUPPLIERS.STATE = "Illinois"
   & SUPPLIERS.SUPPLIER# = supplier#)
```

The result returned by the original query is the aggregate of all names returned by the various instances of this final query.

A query is optimized by first reducing it (by detachment) into a set of irreducible components and then selecting one subquery to use for tuple substitution. That subquery is processed first, and its output is then used as input to tuple substitution. This produces a set of subqueries which may be

further reducible, and the process iterates. Effective optimization depends upon selecting the proper subquery to use for tuple substitution, and this in turn requires some estimate of the cost of processing each of the subqueries. Wong and Youssefi suggest three possible methods for obtaining these estimates. Two involve an assumption of independence among the relations and their fields. The third suggestion is to sample the individual relations. Unfortunately, no indication is made of how the sampling should be done. As will be pointed out in Chapter IV, simply sampling each relation and running the queries on the samples usually will not give a good estimate of query performance.

Multiple-Site

Some "early" work on the optimization of non-relational queries in a network environment was done by Chu [69] and Levin [74]. They were concerned with the allocation of copies of files to network hosts so as to minimize the total database usage cost: the cost of storing files plus the cost of network data transmissions. They presented cost functions based upon the assumption that query traffic generated at each host was known beforehand, and described tractable methods for finding an optimal assignment. Their results are not applicable to the problem addressed in this thesis for two reasons. Theirs is a file allocation problem, whereas this thesis is concerned with the optimization of queries after the files have already been allocated to network hosts. More important, however, is the fact that the queries used in their models were simple. The query

used by Levin was a bit more complex than that used by Chu. It consisted of a message to one copy of a program. The program would in turn send retrieval requests to one copy of a single file. (Chu's query sent requests directly from the originating host to the host with the file, bypassing any reference to programs.) No single query would result in traffic to more than one file. This model is incapable of modelling the complex interactions between files (relations) which will occur in a distributed, relational database.

The more reasonable approach used in SDD-1 (the System for Distributed Databases under development by Computer Corporation of America) was described by Wong [77]. He treats distributed query optimization with a modified form of query decomposition. Basically, Wong's approach is to define three different "views" of the database. As before, the "user view" is the cartesian product of all relations in the database. The "distribution view" consists of one relation per host, each containing the cartesian product of all of the relations stored at that host. The "local view" is the individual relations stored at an individual site. Wong uses a modified form of query decomposition to optimize the query with respect to the distribution view. This modified decomposition only uses tuple substitution (which manifests itself in this case as a series of moves of relations or pieces of relations between network nodes) and the process of splitting off single-relation (i.e. single-host) pieces of the query. This yields a set of subqueries, each of which references only relations local to one host, and a list

of moves of intermediate operands (the outputs from individual subqueries) between network hosts. Each local subquery can then be optimized using normal query decomposition.

This technique suffers from the same problems as single-site decomposition. It is dependent upon some estimate of the sizes of the outputs from individual subqueries. I have discussed above the inadequacies of the techniques in [Wong and Youssefi 76] for making these estimates. Wong's technique also does not allow multiple copies of individual relations. (SDD-1 actually supports multiple copies, but an individual user only has access to a single copy at any one time.) This unnecessarily restricts the scope of application of the technique.

IV. SAMPLING

In this chapter, a new approach to distributed query optimization is described. The concept of sampling the component parts of the database to forecast the operation of a query is discussed in detail. Chapter V gives one method by which the results of such sampling can be used to find an optimal query strategy.

Typically, authors who have needed to know the sizes of intermediate operations have made an assumption of independence between the domains (fields) in the database. (See, for instance, [Wong 77].) Using sampling to make these predictions has one major advantage over the independence assumption: it is possible to get a theoretically valid bound for the error in the estimate. There is therefore some basis for assuming that the estimates are reasonably correct. There is no a priori indication that estimates arrived at using an independence assumption are accurate. In fact, database domains are often not independent. In a personnel file, for instance, one would expect a strong correlation between such fields as "Job Title", "Years of Service", and "Salary".

Several different query types of increasing complexity will be discussed below. Two fairly simple queries will be covered first to introduce the concepts of sampling. Finally, a "generalized" sampling method will be discussed which will allow optimization of the archetypical query described in Chapter II.

Sampling for Simple Select Functions

The simplest query to be discussed consists of a select function for each of two relations and a binary operator (which could be, but need not be, a join) to be performed on the results of the selects. It is assumed that the relations in question are stored on different host computers, so at least one of the intermediate results must be shipped over the net. It is further assumed that it makes no difference where the final result is produced, so the choice of where to do the binary operation depends only on the volumes of the intermediate results. To give an example, the model query looks something like $(R|B_R) * (S|B_S)$. The notation $(R|B_R)$ specifies a Boolean select function B_R to be applied to relation R , and the symbol $*$ denotes a join. If relation R is stored on host 1 only and relation S is on host 2 only, then the output of $(R|B_R)$ should be shipped to host 2 if and only if the volume (defined as the number of bytes in a tuple multiplied by the number of tuples) of $(R|B_R)$ is less than the volume of $(S|B_S)$. This is written as $|R| < |S|$, where $|R|$ denotes the volume of relation R . From now on, for notational simplicity, it is assumed that all tuples are the same size, and the volume is taken to be the number of tuples. A tuple size factor could be easily added if necessary.

Sampling a relation. What is needed is an estimate of P , the proportion of tuples in a relation which satisfy the select function. The approach made here is to take a random sample of the tuples in the relation and find p , the proportion of tuples in the sample which satisfy the select function. It should be

obvious that p is an estimate of P , but how good is it? It seems reasonable to assume that a larger sample will give a more accurate estimate, but a quantitative error estimate would be desirable. For sampling "without replacement" (that is, no tuple can appear in the sample more than once) Yamane [67 p. 98] gives a formula which relates p , the sample size n , the population size N (i.e. the number of tuples in the relation), and the precision d . Omitting the lengthy derivation, the formula is

$$d^2 = \frac{(N-n)z^2p(1-p)}{nN} \quad (1)$$

The positive constant z is a reliability factor which is derived from the normal distribution function. It is related to the desired "confidence level" k by the formula

$$k = F(z) - F(-z) = 2F(z) - 1.$$

$F(x)$ is the cumulative normal probability function

$$F(x) = \int_{-\infty}^x f(y, \theta, l) dy$$

where $f(y, m, s)$ is the probability density function for the normal distribution with mean m and standard deviation s . For a confidence level of 95% ($k = 0.95$), z is 1.96. A z of 3 will give a confidence level of 99.7%. A confidence level of 95% (for instance) means that $|P-p|$ can be expected to be less than d 95% of the time.

It should be noted here that this and later formulas for the precision d are not exact, but rather are "unbiased"

estimates. To get an exact value for d generally requires an exact value for some parameter (p in this case) which could only be obtained by searching the entire database. In this case, using P in place of p in (1) would yield an exact value for d . This is clearly impractical, given that P is the quantity being estimated in the first place. The fact that the formula yields an inexact figure, the precision of which is not computed, is generally not bothersome; it is a second-order effect.

Example: Start with a relation containing 10^5 tuples. After taking a random sample of 500 tuples, it is found that 50 satisfy the select function. Therefore, $p=0.1$. Evaluating (1) (using $z=1.96$) gives $d=0.026$. This means that P will be between 0.074 and 0.126 95% of the time. (This phrasing may be a little misleading. P is a constant, and p is an estimate of it. What is really involved is the confidence that the estimate is accurate within precision d .) Therefore, it would be reasonable to expect $10,000 \pm 2600$ tuples from the full relation to satisfy the select function.

In a practical case, it will be useful to know how large the sample should be to give the desired precision. Equation (1) can be easily inverted to give

$$n = \frac{Nz^2 p(1-p)}{Nd^2 + z^2 p(1-p)} \quad (2)$$

Since n depends on p , it will be necessary to take a small sample of the relation to get a provisional value for p . This provisional p can be used to find the necessary n . If this n is

greater than the original sample size, then the sample must be enlarged. This new sample will yield a new value for p , which will give a new value for n . The process can be repeated until the desired precision is obtained. When constructing a semi-permanent sample, $p=0.5$ will give a worst-case value for n . Yamane says that if n is larger than $N/2$, the actual precision will be greater than that predicted by (1).

Table 1 was adapted from [Yamane 67]. It shows the required sample sizes for various population sizes and precisions.

Experimental results. In order to gain some empirical evidence for the value of sampling, some experiments were run on a test database. This database contains one relation with 10,000 tuples of information on fuel storage at military sites. Twenty-nine queries were run against the whole database and against two different samples, with the results presented in Table 2. Column 1 indicates the number of tuples actually produced by the query when it was run against the whole database. Column 2 gives the query output volume estimated from a simple random sample of 100 tuples, with a precision computed from formula (1). The final column is an estimate of query output volume based upon "classical" independence assumptions. (The method used to obtain the figures in column 3 will be described later in this chapter.) In most cases, sampling yielded an estimate which was accurate to well within the theoretical limits. In addition, the estimates obtained through sampling were generally more accurate than those obtained from classical

Table 1 - Sample size for specified confidence limit and relative precision, in percent, for $p = 0.5$.^a

Population Size	95% confidence interval					99.7% confidence interval				
	Sample size for precision ^c of					Sample size for precision ^c of				
	+1%	+2%	+3%	+4%	+5%	+1%	+2%	+3%	+4%	+5%
500	b	b	b	b	217	b	b	b	b	b
1000	b	b	b	375	277	b	b	b	b	473
1500	b	b	623	428	305	b	b	b	725	562
2000	b	b	695	461	322	b	b	b	825	620
2500	b	1224	747	484	332	b	b	1249	900	661
3000	b	1333	787	500	340	b	b	1363	957	692
3500	b	1424	817	512	346	b	b	1458	1003	715
4000	b	1500	842	521	350	b	b	1538	1040	734
4500	b	1565	862	529	353	b	b	1607	1071	750
5000	b	1622	879	535	356	b	b	1666	1097	762
6000	b	1714	905	545	361	b	2903	1764	1139	782
7000	b	1787	925	552	364	b	3118	1842	1171	797
8000	b	1846	941	558	366	b	3302	1904	1196	808
9000	b	1895	953	562	368	b	3461	1956	1216	818
10000	4899	1936	964	566	369	b	3600	1999	1232	825
15000	5855	2069	996	577	374	b	4090	2142	1285	849
20000	6488	2143	1013	582	376	b	4390	2222	1313	861
25000	6938	2190	1023	586	378	11842	4591	2272	1331	868
50000	8056	2290	1044	593	381	15517	5056	2380	1367	884
100000	8762	2344	1055	596	382	18367	5325	2439	1386	891
→ ∞	9604	2401	1067	600	384	22500	5625	2499	1406	900

^aThis is the proportion of units in the sample possessing the characteristic being measured; for other values of p , the required sample size will be smaller.

^bIn these cases a sample of 50% of the universe will give more than the required accuracy. The formula used in this calculation does not apply when n is more than 50% of N .

^cThis is the precision in percent of the population size, not in percent of p . If $p=0.5$ and the precision is $\pm 5\%$, then $P=0.5 \pm 0.05$.

Table 2 - Two different sampling techniques as estimators of query performance.
 Precesions are at 95% confidence level.

Query	Actual Volume Produced	Estimated From 100 Random Tuples	Estimated From 100 Random Locations	Estimated From Classical Assumptions
state=98 (Means a ship)	1752	1900±765	1755±626	109 ^e
state=uk	190	400±382	93±176	109 ^e
state=06	701	700±498	994±731	109 ^e
fuel=145	1015	600±463	1196±338	400 ^e
fuel=jp4	1227	1400±677	1398±355	400 ^e
fuel=jp4 fuel=145	2242	2000±780	2594±620	800 ^e
command=sac	464	300±333	186±352	86 ^e
command=mac command=sac	607	700±498	419±560	172 ^e
fuel=jp4 & command=mac	27	0±194 ^a	47±88	18 ⁱ
reciepts_dod>100000	9	0±194 ^a	31±59	?
stock<10000	8268	8300±733	7812±1167	?
state=98 & fuel=145	55	100±194	124±123	177 ⁱ
state=98 & fuel=jp4	4	0±194 ^a	16±29	215 ⁱ
fuel=jp4 & stock<10000	638	900±564	668±261	1014 ⁱ
stock≤5000 & stock>0	5130	5400±972	4923±1250	?
stock<5000 & stock>0 & state=98	55	100±194	62±92	899 ⁱ
open_inventory<stock	1953	1800±749	2019±538	?
open_inventory>stock & command=sac	239	100±194	109±205	90 ⁱ
state=98 & stock<10000	1608	1800±749	1460±493	1449 ⁱ
state=98 & stock<10000 & fuel=jp4	3	0±194 ^a	16±29	178 ⁱ
receipts_dod>receipts_comm	1528	1600±715	1600±452	?
command=sac & command=mac & fuel=jp4	1110	1400±677	1305±346	1153 ⁱ
location>us	2671	2300±821	2794±890	?
location>us & state=98	1740	2000±780	1739±627	467 ⁱ
receipts_dod>10	1554	1600±715	1584±472	?
receipts_comm>10	2505	2500±844	2252±665	?
receipts_comm>10 & receipts_dod>10	224	0±194 ^a	248±203	389 ⁱ
command=pac/ships command=pacflt	663	300±333	373±236	172 ^e
(command=pac/ships command=pacflt) & state=98	583	300±333	311±225	116 ⁱ

^aActually, since p=0, this should be 0±0. We give a more conservative precision based upon p=0.01.

^eThis is based on the assumption that each distinct domain value is equally probable.

ⁱThis is based upon independence between domains.

assumptions.

Probability of correct guess. Having gotten p_R , d_R , p_S , and d_S for the two relations in the query, one can decide which intermediate result to ship over the net. It would be useful to know the probability that the choice made is, in fact, optimal. If $|R|$ and $|S|$ are the volumes of the relations R and S , then the estimates of $|(R|B_R)|$ and $|(S|B_S)|$ are $p_R|R|$ and $p_S|S|$. Assume that $p_R|R| > p_S|S|$. The best strategy then is to ship the relation $(S|B_S)$. The exact values of $|(R|B_R)|$ and $|(S|B_S)|$ will usually be different from the predicted values, but as long as $|(R|B_R)| \geq |(S|B_S)|$, the correct decision will have been made.

The confidence in the decision is merely the probability that $|(R|B_R)| \geq |(S|B_S)|$ is true. If it is assumed that the two values are normally distributed around the estimated values, then it can be shown that the "probability of correct guess", if relation $(S|B_S)$ is shipped, is

$$F\left(\frac{(p_R|R| - p_S|S|)z}{\sqrt{(d_R|R|)^2 + (d_S|S|)^2}}\right) \quad (3)$$

A derivation of this is given in Appendix A. Sample values of this function, for $d_R|R| = d_S|S|$, are presented in Table 3. For instance, suppose $|R| = |S| = 10^4$, $p_R = 0.1000$, $p_S = 0.084$, and $d_R = d_S = 0.02$, then $d_R|R| = d_S|S| = 200$, and $p_R|R| - p_S|S| = 1000 - 840 = 160$. From the table, one can deduce that by shipping the output from S , one will have made the correct decision 86.6% of the time.

Expected excess cost. From (3), it is possible to find the

		$p_{R R_i} - p_{S S_i}$					
		0	40	80	120	160	200
$d_{R R_i} = d_{S S_i}$	20	0.500	0.997	1.000	1.000	1.000	1.000
	40	0.500	0.917	0.997	1.000	1.000	1.000
	60	0.500	0.822	0.968	0.997	1.000	1.000
	80	0.500	0.756	0.917	0.981	0.997	1.000
	100	0.500	0.710	0.866	0.952	0.987	0.997
	120	0.500	0.678	0.822	0.917	0.968	0.990
	140	0.500	0.654	0.786	0.883	0.943	0.976
	160	0.500	0.636	0.756	0.851	0.917	0.958
	180	0.500	0.621	0.731	0.822	0.891	0.938
	200	0.500	0.609	0.710	0.797	0.866	0.917
	220	0.500	0.599	0.693	0.775	0.843	0.896
	240	0.500	0.591	0.678	0.756	0.822	0.876
	260	0.500	0.584	0.665	0.739	0.803	0.857
	280	0.500	0.578	0.654	0.724	0.786	0.839
	300	0.500	0.573	0.644	0.710	0.770	0.822
	320	0.500	0.569	0.636	0.698	0.756	0.807
	340	0.500	0.565	0.628	0.688	0.743	0.793
	360	0.500	0.561	0.621	0.678	0.731	0.779
	380	0.500	0.558	0.615	0.669	0.720	0.767
	400	0.500	0.555	0.609	0.661	0.710	0.756

Table 3 - Probability of correct guess
when relation S is shipped

		$p_{R R_i} - p_{S S_i}$					
		0	40	80	120	160	200
$d_{R R_i} = d_{S S_i}$	20	5.76	0.01	0.00	0.00	0.00	0.00
	40	11.51	1.09	0.02	0.00	0.00	0.00
	60	17.27	4.16	0.55	0.04	0.00	0.00
	80	23.03	8.35	2.18	0.40	0.05	0.00
	100	28.79	13.10	4.87	1.44	0.33	0.06
	120	34.54	18.17	8.32	3.27	1.09	0.31
	140	40.30	23.42	12.32	5.81	2.44	0.90
	160	46.06	28.80	16.69	8.92	4.37	1.95
	180	51.82	34.25	21.34	12.48	6.82	3.47
	200	57.57	39.77	26.20	16.40	9.73	5.46
	220	63.33	45.33	31.21	20.61	13.03	7.86
	240	69.09	50.92	36.33	25.04	16.64	10.65
	260	74.85	56.54	41.55	29.65	20.52	13.76
	280	80.60	62.18	46.84	34.41	24.63	17.16
	300	86.36	67.83	52.19	39.30	28.93	20.80
	320	92.12	73.50	57.59	44.28	33.39	24.66
	340	97.87	79.17	63.03	49.35	37.98	28.71
	360	103.63	84.86	68.51	54.50	42.69	32.91
	380	109.39	90.55	74.01	59.70	47.50	37.26
	400	115.15	96.25	79.54	64.96	52.40	41.73

Table 4 - Expected excess cost, in tuples,
when relation S is shipped

likelihood of making the wrong decision. It is also useful to know how much will be lost as a result of the fact that the choice made is not always correct. The expected excess cost is the expected number of extra tuples that will be shipped if the wrong decision is made, multiplied by the probability that the wrong decision will be made. When relation $(S|B_S)$ is shipped, this can be computed from the formula

$$\int_{-\infty}^{\infty} \int_x^{\infty} (y-x) f(x, p_R |R|, \frac{d_R |R|}{z}) f(y, p_S |S|, \frac{d_S |S|}{z}) dy dx.$$

This is the expected value of $(y-x) = |(S|B_S)| - |(R|B_R)|$ computed over all x and y such that $y > x$. This formula can be simplified to

$$a \int_b^{\infty} (1-F(x)) dx$$

where

$$a = \frac{\sqrt{(d_R |R|)^2 + (d_S |S|)^2}}{z}$$

and

$$b = \frac{p_R |R| - p_S |S|}{a}$$

Sample values of expected excess cost are presented in Table 4. The table entries are the expected number of extra tuples that will be shipped, for given precisions and differences in expected volumes. In the example for Table 3, where $d_R |R| = d_S |S| = 200$ and $p_R |R| - p_S |S| = 160$, one would expect, on the average, to ship 9.73

tuples more than are necessary. (The smaller relation will actually be shipped 86.6% of the time, and 13.4% of the time an average of $9.73/.134=72.61$ extra tuples will be shipped.)

Join Prediction

The above random sampling technique will allow prediction of the volume of output from a restrict on a relation. Unfortunately, optimizing more complex queries involving several levels of joins will require estimating the volume of output from join operations. The following sections discuss a query consisting of two restricted relations joined on a single domain. In this case, it is assumed that the output from the join might have to be shipped. It is therefore necessary to estimate the volume of output from the join in addition to the volume of output from each select.

For this query, there are several strategies which could be used, depending on the relative volumes of the database relations. There are two cases which should be considered: both relations large and one large and one small relation. Each case will be considered separately.

One large and one small relation. If one relation is very much larger than the other, it will be worthwhile to use sampling on only the larger relation. The three different volumes involved are computed using different methods. The volumes and the method used for predicting each are as follows:

Result of select on smaller relation: This volume should

be found by simply running the select on the entire small relation. The relation is assumed to be small enough that the cost of performing the select and/or shipping its output more than once is minimal compared to the total query cost.

Result of restrict on larger relation: This volume should be predicted using simple random sampling. The method used is the same as that used for the two volumes in the first query considered.

Result of the join: Let X be the number of tuples that will actually be produced by the join. This volume can be predicted by estimating the average number of tuples in the join output which come from each tuple in the larger relation. This can be done by joining the sample of the larger relation with the smaller relation after performing the respective select functions. If the sample of the larger relation contained n of the total N tuples, and this "sample join" yielded x tuples, then the sample tuples produced an average of x/n tuples each in the join. Therefore, a reasonable estimate for X is $\hat{X} = Nx/n$. If x_j is the number of tuples in the sample join which come from the j th sample tuple in the larger relation (note that $\sum_j^n x_j = x$), then the precision of the estimate of X can be computed from Yamane's equation for the precision of the "total of a population". Again, sampling without replacement is assumed, where no tuple from the larger relation can appear in the sample more than once. His formula (from page 84) is

$$d^2 = z^2 N^2 \frac{s^2}{n} \frac{(N-n)}{N}$$

where

$$s^2 = \frac{\sum_{j=1}^n (x_j - \frac{x}{n})^2}{n-1}$$

As before, if k is the confidence level associated with the given z , then $|X-\bar{X}|$ will be less than d k percent of the time.

Both relations large. When attempting to model the join between two large relations, one might be tempted to take a simple random sample of each relation, form the join of the samples, and use this sample join to model the actual join. Unfortunately, because each sample will contain only a small fraction of the possible joining domain values, this sample join will not "mesh" the same as the actual join. Consider, for instance, a join between two relations of 10,000 tuples each, where the joining domain is a key for each relation consisting of the integers from 1 to 10,000. If 1% samples of 100 tuples are taken from each relation and then joined together, only one tuple could be expected to be found in the sample join. This is even without any select functions being performed. The join of the two unrestricted relations would produce 10,000 tuples, so this is a sample of size 1 from a population of 10,000. This will give a poor precision, indeed. What is needed is a sampling technique which will estimate the output from each relation and also estimate the output of the join.

Toward this end, a variation of the sampling technique is

introduced here. In previous discussions the basic sampling unit was a single tuple. Instead, take individual values of the joining domain to be the basic sampling unit. Before, the output volume was estimated by estimating the probability that each tuple would be selected for output, and multiplying this probability by the total number of tuples. Instead, the volume can be estimated by first estimating the average number of tuples with each domain value and multiplying by the number of distinct domain values. If m of the M possible domain values are picked as the sample of the domain, and n_j is the number of tuples with the j th sampled domain value which satisfy the output conditions, then the total number of tuples in the output can be estimated as

$$\hat{N} = \frac{M}{m} \sum_j^m n_j$$

The precision of this is found from

$$d^2 = z^2 M^2 \frac{s^2}{m} \frac{(M-m)}{M} \quad (4)$$

where

$$s^2 = \frac{\sum_j^m (n_j - \frac{\hat{N}}{M})^2}{m-1}$$

This equation for d can be easily inverted to get the sample size m required to produce a given precision when the sample variance is s^2 .

To implement this method for predicting join output, first take a sample of the joining domain values. From each relation

select and save all tuples which have one of these values. The query in question should be run against these samples. The above formulas can then be used to predict the performance of the query when run against the full relations.

There are, of course, disadvantages to this scheme. If the multiplicity (number of tuples with a given joining domain value) and the variance (s^2 in the above equations) are not small, then the number of tuples required for a given precision will be larger than for simple random sampling. Also, because constructing a sample will require searching the relation instead of just picking random tuples, the cost of constructing the sample will be much higher. (The presence of an appropriate index would reduce this extra cost factor.)

Another problem inherent in the scheme has to do with the fact that the sample is no longer truly random. In particular, if a select function references the joining domain explicitly, the prediction could be off. There are two ways of coping with this problem. It could be ignored (with some justification) in the hope that the sample will be large enough to smooth out this effect. Alternatively, the query processor could be made smart enough to separate the select into parts that do not depend on the joining domain. These parts could be run against the sample separately and the results combined using an independence assumption. For instance, suppose the join is on domain "location" and the query is "(location = 'London' & stock > 1000) or (location = 'Paris' & stock > 5)." If the sample shows that the "average" location has five tuples with stock > 1000 and ten

with stock > 5, then one can expect 15 tuples to result from this query. It will sometimes happen that the domain value(s) tested in the query will be included in the sample. In this case it is not necessary to look at the full relation at all. If, in the above example, both "London" and "Paris" were in the sample, the query could be processed using only the sample. (In light of this, it is tempting to place the most frequently referenced domain values in the sample. This would have to be done with extreme care, however, to preserve the statistical properties of the sample.)

Experimental results. A second sampling method was used on the same test database as before. This was a sample by domain value. One hundred of the 1553 values of the field called "location" were selected, and the 663 records which had one of those values were collected. The predicted volumes in column 3 of Table 2 are 155.3 times the volumes from the queries run on this sample. The precisions are obtained from equation (4).

Generalized Query Sampling

All of the above discussions have dealt with a very simple query operating on only two relations. This section describes a sampling technique which is applicable to more complex queries consisting of joins and restricts on an arbitrary set of relations. By constructing specially designed samples of the individual relations, this technique will allow estimation of the volume of output from each intermediate result in the query.

Unfortunately, it is difficult in this general case to define the probability of correct guess and expected excess cost described in earlier sections. These statistics were defined for a simpler, two-relation query for which there were only two possible query strategies. It was easy to treat the tradeoffs between the two strategies. In the current, more general case, there could be a very large number of feasible query strategies for any given query. The probability of correct guess in this case would be the probability that the chosen strategy is better than all alternatives. To compute this would require some kind of enumeration of the possible strategies and a computation of the effect of the errors in the estimated volumes on the cost of each strategy. Furthermore, in the two-relation case each relation was sampled independently, so it was reasonable to assume that errors in the volumes being estimated were independent. This made it easy to treat the problem analytically. In the general case, the relation samples are not built independently, so it is unreasonable to assume independence of the errors. For these reasons, no attempt has been made to define probability of correct guess or expected excess cost in this context.

A Multiple-Relation Sampling Technique. Consider now an arbitrary relational-algebra query to the database. This query can be represented as a tree. Each leaf node represents a select function to be applied to some stored relation. Each interior node represents a join between two relations and a select on the result of the join. This tree can be locally optimized using a

procedure, similar to that described by Smith and Chang [75], which "moves down" portions of the select functions as far as possible in the tree. This will cause tuples to be removed as early as possible, at low levels in the tree, thus reducing the total volume of intermediate results. To optimize the assignment of tree nodes (query operations) to network hosts, it is necessary to be able to estimate the volume of output from each operation in the query tree.

The approach developed here is an extension of one described in an earlier section. For predicting the output of a join between two large relations, the suggested approach was to select a sample of the domain values and estimate the number of tuples in the relation which have each value. To predict arbitrary queries, we will select one database domain as a "base" domain B from which sample values will be taken. Samples of individual relations will be constructed which consist of all tuples which either contain one of the sampled B values or which could be joined, through some series of join operations, with a tuple containing one of the sampled B values. The assumption of join uniqueness ensures that the relation samples will be unique for any given base domain sample.

The procedure described below selects tuples to be included in the sample by first identifying samples of some domains which could be involved in query joins. Every database relation will contain at least one domain which has been sampled. The samples of the individual relations will consist of all tuples which contain one of the sampled domain values. For each sampled

domain D in the database, the sets S_{Di} and the constant z_D will be developed. S_{Di} is the set of all values of domain D which are "associated with" the sampled value b_i of the base domain B . The values b_i of B and d_j of D are defined to be associated with each other if there exists a tuple in GOAL which contains those two values. The constant z_D is the number of joining steps needed to get from the relation containing B to the relation containing D .

1. First, the base domain B must be selected. This could be any domain in the database, but for best results the joining domain with the largest number of distinct values or a key domain of GOAL should probably be selected. From the M distinct values of B , select a random sample of m values. For each sampled value b_i , the set S_{Bi} contains only the value b_i . The "distance" z_B is equal to zero.
2. Let T be any relation in the database which contains no domains for which samples have been constructed, but which can be joined to relation R by domain D , where R contains a domain C which has been sampled. If no such T exists, the sampling procedure is finished.
3. Use the notation $(c_j, d_k) \in R$ to indicate that a tuple with a domain C value of c_j and a domain D value of d_k exists in R . The sets S_{Di} can be constructed as

$$S_{Di} = \{d_k \mid c_j \in S_{Ci}, (c_j, d_k) \in R\}.$$

S_{Di} is the set of all D values which appear in R together with one of the C values in S_{Ci} . The values in S_{Di} are

therefore all of those associated with values in S_{C_i} .

The domain C is either B itself or it is one of a set of domains which can be used to join R (possibly thru a series of intermediate linking relations) to a relation containing B. The nature of the join operation ensures that if any R tuple with C value c_j can be joined to a tuple with B value b_i (again, possibly using intermediate joins), then every R tuple with value c_j will be joined to a tuple with value b_i . Since the join operation must reconstruct pieces of GOAL, and since S_{C_i} contains all tuples associated with b_i , it follows that S_{D_i} contains all D values associated with b_i . Conversely, a value d_k cannot appear in S_{D_i} unless it is associated with b_i , so S_{D_i} is exactly the set of D values associated with b_i .

Having identified the sample of D, the sample of T can be constructed consisting of all those T tuples which have D values appearing in one of the sets S_{D_i} . If there are any other relations in the database containing domain D and which have not yet been sampled, their samples can also be constructed from the S_{D_i} .

4. Let $z_D = z_C + 1$.

Go to step 2.

The sample of the database consists of the individual relation samples developed above.

Each individual value of a sampled domain may be associated

with several values of B . The probability that a given domain value will be included in the sample is directly proportional to the number of B values it is associated with. Different domain values will therefore have different probabilities of being included in the sample. If this effect is not compensated for, the estimates will be biased in favor of tuples with values which are associated with a large number of B values. Therefore, the relation samples must be used with appropriately developed weights which will cause "high probability" tuples to have proportionally lower impact on the estimates.

For each value d_j which appears in one of the sets S_{D_i} , a weight m_{D_j} must be defined. This weight is equal to the number of distinct B domain values which d_j is associated with. The computation of these weights is by far the most expensive portion of the setup cost of this sampling method. The only way to find them in the general case is to enumerate the base domain values which could be joined to each sampled value in the sampled domain D . This could be done by finding the set S'_{D_j} , the set of B values associated with sampled value d_j , in a manner similar to that used to find S_{D_i} . Alternatively, a sample of GOAL could be constructed containing all tuples with a sampled domain D value, using appropriate joins of individual relations. The m_{D_j} values could be computed directly from this relation.

There is a special case where computation of m_{D_i} becomes somewhat easier. This is when the base domain B is a key of GOAL. One such key can always be constructed, if necessary, by

combining several domains.¹ In this case, for the relation R and domains C and D (from step 2 above),

$$m_{Dk} = \sum_{\substack{j \\ (c_j, d_k) \in R}} m_{Cj}.$$

In other words, the weight on the domain value d_k is equal to the sum of the weights on the domain C values which are associated with d_k . The weights on all values of the base domain are 1. The weights for the other domains can be computed as part of the procedure defined above. At each step, the weights for all values of a domain D (not just the values in the sample) can be computed from the weights on C.

To estimate $|T|$, the volume of a relation T which is the output of some query or piece of a query, run the query on the database sample. Call T' the result of running the query on the relation samples, and let D be a sampled domain in T such that $z_D \leq z_A$ for all sampled domains A in T. If x_j is the number of tuples in T' with a domain D value of d_j , M is the number of distinct base domain values and m is the number of them that are in the sample, then $|T|$ can be estimated as

¹If B is a composite domain, the sampling procedure must treat B as distinct from its components. For instance, if B is the composite of domains A and C (call it AC), then the procedure could generate samples of A or C or both in addition to the sample of AC. If, in addition, no stored relation exists which contains all of the domains of B, it will be necessary to construct a temporary relation containing only B for sampling purposes. It can be discarded when the sampling parameters have been computed.

$$\bar{N} = \frac{M}{m} \sum_i n_i$$

where

$$n_i = \sum_{d_j \in S_{Di}} \frac{x_j}{m_{Dj}}$$

The quantity n_i is a weighted count of the number of tuples in T' with D values associated with B value b_i .

As mentioned above, probability of correct guess and expected excess cost will not be computed for this type of query. However, the accuracy of any intermediate volume estimate can be obtained as before from the formula

$$d^2 = \frac{z^2 M(M-m)}{m} s^2$$

where

$$s^2 = \frac{\sum_i (n_i - \frac{\bar{N}}{M})^2}{(m-1)}$$

Example. Consider a database consisting of four relations. It contains information on parts and suppliers who supply those parts. Relation PARTS contains information for each part, relation SPLR contains information for each supplier, relation CITIES contains information on individual cities in which suppliers are located, and relation SP indicates which suppliers supply which parts.

PARTS		SP		SPLR			CITIES	
PNAME	P#	P#	S#	S#	SNAME	City	City	State
Widget	1	1	1	1	Ajax	Urbana	Flagstaff	NM
Bolt	2	2	1	2	Acme	Hinkley	Hinkley	OH
Nut	3	2	3	3	Widget	Urbana	Urbana	IL
Rivet	4	1	4	4	Bomad	Flagstaff		
		3	2					
		3	4					
		4	4					
		4	2					

Suppose that the composite domain $(P\#,S\#)$ is selected as the base domain. (This is a key of GOAL.) Call this composite domain C , and let the sample of C be the values $(1,1)$, $(2,3)$, and $(1,4)$. The domain samples generated from this sample of $(S\#,P\#)$ are

$$\begin{array}{llll}
 S_{C_1} = \{(1,1)\} & S_{P\#,1} = \{1\} & S_{S\#,1} = \{1\} & S_{City,1} = \{Urbana\} \\
 S_{C_2} = \{(2,3)\} & S_{P\#,2} = \{2\} & S_{S\#,2} = \{3\} & S_{City,2} = \{Urbana\} \\
 S_{C_3} = \{(1,4)\} & S_{P\#,3} = \{1\} & S_{S\#,3} = \{4\} & S_{City,3} = \{Flagstaff\}
 \end{array}$$

The weights resulting from this choice of base domain are

P#	$m_{P\#}$	S#	$m_{S\#}$	City	m_{City}
1	2	1	2	Urbana	3
2	2	2	2	Hinkley	2
3	2	3	1	Flagstaff	3
4	2	4	3		

To test the effect of a given query which uses only the SPLR relation, for instance, run the query on the sample relation SPLR' which contains only tuples with $S\#$ values of 1, 3, or 4. If that query selected only the tuple $(4, Bomad, Flagstaff)$, then an estimate of the actual output from the whole SPLR relation would be:

$$\hat{N} = \frac{8}{3} \left(\frac{0}{2} + \frac{0}{1} + \frac{1}{3} \right) = 0.89$$

The precision of this at the 95% confidence level is:

$$d^2 = \frac{(1.96)^2 (8) (5) ((0-0.11)^2 + (0-0.11)^2 + (0.33-0.11)^2)}{(3) (2)}$$

$$= 1.86$$

$$\Rightarrow d = 1.36$$

Prediction of Project Output

In the discussion up to this point, the project operation has been deliberately ignored. This is because not all sampling methods can be used to predict the effect of a project. The project operation will coalesce several input tuples into one output tuple, so if the sampling method used was such that some tuples in the sample could possibly be merged with tuples not in the sample, it would be impossible to accurately predict the output volume. There would be no way to estimate from the sample alone how many input tuples would be projected into a single output tuple.

It turns out, however, that the multiple-relation sampling technique just described will handle project in the majority of cases. As long as a given project retains at least one of the sampled domains, it will be possible to predict the output of the project using that sampled domain. As before, the query, including the project, is run on the sample database, and then the output volume is predicted using the sampled domain. This will work because the sample database, by definition, contains

all tuples which have any one of the sampled domain values. Hence, there can be no tuples not in the sample which could be merged with a tuple not in the sample. Since all joining domains are sampled, there will always be a sampled domain in the output of any project which will later be the input to a join. Therefore, all volumes (except sometimes the final output volume) in a query with projects can be predicted. In the cases where the final output volume cannot be predicted using sampling, any gross estimate of the volume can be used to find a sub-optimal strategy.

Pathological Cases

It is quite possible that any given database will exhibit pathological behavior which will make it impractical to implement sampling for the entire database. The samples required for some of the relations might prove to be substantial portions of the relations themselves. In this case, the cost of sampling for those relations will be an unacceptably large proportion of the total query cost for that relation. There are at least two situations in which this can occur.

Wide variation in relation sizes. If there is a very wide range in the sizes of the relations of the database, it is probable that the smaller relations will have samples which are appreciable proportions of the whole relations. To see how this happens, consider a three-relation database which is an upgraded version of the two-relation one discussed earlier in this chapter.

<u>Relation</u>	<u>Domains</u>	<u>Number of Tuples</u>
SUPPLIERS	Name, Splr#	1000
PARTS	Color, Shape, Part#	100,000
S-P	Splr#, Part#	300,000

The SUPPLIERS relation has one tuple to describe each supplier, the PARTS relation has one tuple identifying each part, and S-P is a "linking" relation which indicates which suppliers supply each part. On the average, each supplier supplies 300 different parts, and each part is supplied by 3 suppliers. Suppose that Part# is selected as the "base" domain, and 1000 of its 100,000 values are sampled. There will then be 1000 tuples in the sample of PARTS, or 1% of the whole relation. Because each part number (value of Part#) appears about 3 times in S-P, it is reasonable to expect about 3000 tuples in the sample of S-P, which is still only 1% of the tuples in that relation. However, because each Part# value is associated with a large number of Splr# values, it should happen that a large percentage of the SUPPLIERS tuples (over 95% based upon strictly probabilistic considerations) will be in the sample. This means that the sampling cost for the part of the query which uses SUPPLIERS will be close to the actual processing cost for that part of the query. In this case, sampling should not be used on the SUPPLIERS relation. It is so small, anyway, that it contributes only a small part of the total query cost. Instead, the part of the query which uses this relation should be run on any copy of SUPPLIERS which is available on the network. The rest of the query can then be optimized using exact results for the output from this part of the query, rather than estimates based upon sampling.

Large "fan-out" between domains. The term "fan-out" is used very loosely here to refer to the average number of domain values in one domain associated with each value of some other domain. Even if the relations are of similar size, a large fan-out can cause large samples. In the last example, the Splr# domain has a large fan-out to domain Part#. Suppose Splr# is picked as the base relation with a 10% sample of 100 of its values. The sample of SUPPLIERS would contain 100 tuples. Because each Splr# value occurs about 300 times in S-P, the sample of S-P will have about 30,000 tuples. About 27,100 tuples could be expected in the sample of PARTS, or over 27% of that relations's tuples. (There will be some duplication of the Part# values selected by the sample of Splr#, which is why the total sample of PARTS contains fewer than 30,000 tuples.) A 10% sample of SUPPLIERS therefore results in a 27% sample of the larger PARTS relation. In a large database with many relations, this "fan-out" problem could result in unreasonably large samples. In such cases, a more judicious choice of base relation might help, or it might be necessary to eschew sampling for those relations with unreasonably large samples. This means that the "samples" for those relations would consist of the relations, themselves.

V. AN INTEGER LINEAR PROGRAM FOR OPTIMAL QUERY STRATEGIES

Chapter IV described a method which can be used to estimate the amount of data produced by each operation in the query. In this chapter, a method for generating a query strategy based upon those estimates is discussed. This is an integer linear programming (ILP) model which will produce an optimal assignment of query operations to network hosts. Solving a general ILP is an expensive proposition, so a procedure is developed which will allow much less expensive linear programming (LP) techniques to be used, instead.

This technique is not guaranteed to find the best strategy for the given query. Finding such a global optimum would require considering many possible permutations of the query tree. In particular, the fact that the join operation is associative (within certain limits) would have to be considered. Altering the order in which joins are performed can have a large effect on the total size of the various intermediate operations. However, this procedure will find the optimal strategy based upon the given query tree and estimates of intermediate volumes.

Definition of the Integer Linear Program

The cost of a particular implementation of a relational algebra query in a distributed environment can be expressed with the cost function

$$C = \sum_{qi} \sum (E_{qi} x_{qi} + V_q t_{qi})$$

where

E_{qi} is the expense of performing operation q on host i ,

V_q is the expense of sending the output of operation q over the network (An ARPANET-like cost structure is assumed where cost is determined wholly by volume of traffic.),

x_{qi} is 1 if operation q is performed on host i , and is zero otherwise, and

t_{qi} is 1 if operation q is performed on host i and its output must be shipped over the network, and is zero otherwise.

The constants E_{qi} represent purely local processing costs, and are included in the cost function here only for completeness. No suggestions are made of how to compute them. The constants V_q are the estimated volumes of output from the operations, multiplied by an appropriate cost factor. (The constants E_{qi} will also depend partly on the estimated volumes for the inputs to operation q .) All cost constants are assumed to be positive.

Let there be N operations and M hosts, and denote the successor (or parent) operation of operation q as σ_q . (This means that the output of operation q is input to operation σ_q .) A minimum-cost strategy for servicing the query can be found by solving the integer linear programming problem (ILP)

Find values for x_{qi} and t_{qi} which minimize

$$C = \sum_i (\sum_q E_{qi} x_{qi} + \sum_{q=1}^{N-1} V_q t_{qi})$$

under the constraints

$$\sum_i x_{qi} = 1 \quad q=1, \dots, N \quad (5)$$

$$x_{qi} - x_{\sigma_q i} - t_{qi} \leq 0 \quad \begin{array}{l} q=1, \dots, N-1 \\ i=1, \dots, M \end{array} \quad (6)$$

$$x_{qi}, t_{qi} = 0 \text{ or } 1 \quad (7)$$

The constraints (5) ensure that each operation is performed on exactly one host. The constraints (6) ensure that t_{qi} is one if operation q is performed on host i and σ_q is performed somewhere else. If both q and σ_q are performed on host i , or if q is not performed on host i , then the fact that V_q is positive (by assumption) will ensure that the minimization procedure will produce a t_{qi} of zero.²

It is assumed here that the output of operation N , the last operation in the query, will never be shipped over the network. Therefore, the variables t_{Ni} and the constant V_N are left unused in this formulation. If it became necessary to allow the output of N to be shipped, it would be easy to add a final, dummy

²It would be possible in this ILP formulation to replace the M t_{qi} variables with a single variable t_q . However, this simplification would render the linear programming formulation given below unworkable.

operation $N+1$, where $\sigma_N = N+1$.

Notation

In standard terminology, any assignment of values to all of the variables x_{qi} and t_{qi} is called a solution to the ILP. A solution which satisfies all of the constraints is called a feasible solution. A feasible solution which minimizes the cost function (i.e. has a cost which is less than or equal to the cost of any other feasible solution) is called an optimal solution. In this thesis, a solution is feasible with respect to operation Q if all constraints of type (6) are satisfied when q is a descendant of Q , and all constraints (5) are satisfied when q is either equal to Q or is a descendant of Q .

In the discussion which follows, a solution to the above ILP will be represented as a vector of length $2(N-1)M$. The vector corresponding to a given solution S with variables equal to x'_{qi} and t'_{qi} will be

$$(x'_{11}, x'_{12}, \dots, x'_{21}, x'_{22}, \dots, x'_{NM}, t'_{11}, t'_{12}, \dots, t'_{(N-1)M})$$

Similarly, the cost constants can be put in a vector of the form

$$(E_{11}, E_{12}, \dots, E_{21}, E_{22}, \dots, E_{NM}, V_1, V_1, \dots, V_{N-1})$$

where each volume constant V_q occurs M times. This vector will be called E , so the cost of a particular solution S is SE^T . A solution which has the variable x_{qi} (t_{qi}) equal to one and all others equal to zero will be represented by the vector \hat{x}_{qi} (\hat{t}_{qi}).

The value of the variable x_{qi} (t_{qi}) in a solution

represented by the vector S will be denoted by $S[x_{qi}]$ ($S[t_{qi}]$). The symbol δ_{ij} is the Kronecker Delta, which is equal to 1 if $i=j$ and zero if $i \neq j$.

Solving the ILP as a Linear Program

The above ILP can be treated as a Linear Program (LP) and solved using classical methods (e.g. the simplex method) by changing the constraints (7) to

$$x_{qi}, t_{qi} \geq 0 \quad (7')$$

However, there is no guarantee that an all-integer solution will result. It will be shown here, however, that if there are any feasible solutions, there will always be at least one all-integer solution which is optimal. Given an optimal solution S (and the existence of a feasible solution and the fact that the costs are positive guarantee the existence of at least one optimal solution) the following section will describe how to construct a set of integer, feasible solutions p_i and associated positive weights w_i such that

$$\sum_i w_i = 1$$

and

$$\sum_i w_i p_i = S.$$

Because S is an optimal solution, it is true that $p_i E^T \geq S E^T$ for each p_i . It is also true that $\sum_i w_i p_i E^T = S E^T$. It follows that $p_i E^T = S E^T$ for each p_i , so p_i is an all-integer, optimal solution.

Having proved the existence of an integer, optimal solution, a fast algorithm which will produce an integer, optimal solution from an arbitrary, non-integer, optimal solution will be given. A proof of its validity will also be given.

Construction of the Partial Solutions and Weights. Given an arbitrary optimal solution S , the following paragraphs show inductively how to construct a set P_q for each node q in the operation tree. Each element in the set P_q is a pair in the form $(w;p)$ consisting of a weight w and a solution vector p . The weights and vectors in a given P_q will have the following properties:

$$A: \sum_{(w;p) \in P_q} w = 1$$

i.e. the weights sum to unity.

$$B: \text{If } \sum_{(w;p) \in P_q} wp = T_q,$$

then $T_q[x_{ui}] = S[x_{ui}]$ for all hosts i , where u is either q or one of its descendants.

$$C: T_q[t_{ui}] = S[t_{ui}] \text{ for all hosts } i \text{ where } u \text{ is a descendant of } q.$$

D: Each solution p in P_q is feasible with respect to operation q .

The set P_N , where N is the root node of the query, contains the weights and all-integer, optimal solutions which can be used to reconstruct S as above.

First, construct the sets P_q for each q that is a leaf of the query tree. Each P_q will be

$$P_q = \bigcup_{\substack{i \\ S[x_{qi}] > \emptyset}} (S[x_{qi}]; \hat{x}_{qi})$$

The equations (5) in the LP ensure that the weights in P_q sum to 1, so P_q satisfies property A. The other properties are true trivially.

The next task is to show that if P_r and P_s exist with these properties, where r and s are the children of node q , then P_q can be constructed with the same properties.³ This will be the basis for an inductive proof that such a set can be constructed for the root node N .

Select a node q with children r and s (meaning that $\sigma_r = \sigma_s = q$) such that P_r and P_s have been constructed and P_q has not. Consider the child r . Construct and solve the transportation problem

$$\text{Minimize } \sum_{ij} (1 - \delta_{ij}) g_{ji}$$

under the constraints

$$\sum_j g_{ji} = S[x_{qi}] \quad i=1, \dots, N$$

³It is assumed here that all operations are binary. The extension of this procedure to non-binary operations is straightforward. Each new operand would introduce another transportation problem and two more ranges to "union" over.

$$\sum_i g_{ji} = S[x_{rj}] \quad j=1, \dots, N$$

$$g_{ji} \geq 0$$

The variable g_{ji} is, in a sense, the part of the output from operation r which is generated at host j and used at host i . The quantity minimized can be thought of as the total amount of output from operation r that is shipped over the network. A similar transportation problem will yield n_{ki} for the other child s .

The set P_q is then defined as

$$P_q = \bigcup_{\substack{i \\ S[x_{qi}] > 0}} \bigcup_{\substack{j \\ g_{ji} > 0}} \bigcup_{\substack{(w_r; p_r) \in P_r \\ p_r[x_{rj}] > 0}} \bigcup_{\substack{k \\ n_{ki} > 0}} \bigcup_{\substack{(w_s; p_s) \in P_s \\ p_s[x_{sk}] > 0}}$$

$$\left(\frac{w_r w_s g_{ji} n_{ki}}{S[x_{qi}] S[x_{rj}] S[x_{sk}]}; p_r + p_s + \bar{x}_{qi} + (1 - \delta_{ij}) \bar{t}_{rj} + (1 - \delta_{ik}) \bar{t}_{sk} \right)$$

Proof That the Solutions Satisfy Properties A, B, C, and D.

Let p_q be one of the vectors generated by this procedure from a given p_r and p_s . The vectors p_r and p_s are orthogonal, meaning $p_r[x_{ij}]p_s[x_{ij}] = p_r[t_{ij}]p_s[t_{ij}] = 0$ for all i and j . This is because the subtrees defined by r and s are disjoint. In fact, all of the five vectors summed to form p_q are mutually orthogonal. This guarantees that all of the components of p_q are either 0 or 1. To demonstrate that p_q is feasible with respect to q , note that p_r and p_s are feasible with respect to r and s ,

respectively. Also note that any ILP constraint satisfied by either p_r or p_s will be satisfied by p_r+p_s , so p_r+p_s is feasible with respect to r and s . It must be shown that $\sum_i x_{ui}=1$ for u equal to q or a descendant of q , and $x_{ui}-x_{\sigma_{ui}}-t_{ui}\leq 0$ for u a descendant of q . The induction hypothesis ensures that $\sum_i x_{ui}=1$ for u a descendant of q , and $x_{ui}-x_{\sigma_{ui}}-t_{ui}\leq 0$ for u a descendant of r or s . The inclusion of the term \hat{x}_{qi} in the definition of P_q ensures that $\sum_i x_{qi}=1$. The terms $(1-\delta_{ij})f_{rj}$ and $(1-\delta_{ik})f_{sk}$ ensure that $x_{ui}-x_{qi}-t_{ui}\leq 0$ for u equal to either r or s . Therefore, p_q is feasible with respect to q . This is true for all vectors in P_q , so P_q has property D.

The fact that p_q has property A will be shown as a subresult while demonstrating properties B and C. Properties B and C must be demonstrated in several steps. Let

$$T_q = \sum_{(w;p) \in P_q} wp.$$

First, it will be shown that $T_q[x_{qi}] = S[x_{qi}]$ for any host i . The value of $T_q[x_{qi}]$ can be found by summing the weights on all solution vectors p which have $p[x_{qi}] = 1$. This sum is equal to

$$\sum_{\substack{j \\ S[x_{rj}] > 0}} \sum_{\substack{(w_r;p_r) \in P_r \\ p_r[x_{rj}] = 1}} \sum_{\substack{k \\ S[x_{sk}] > 0}} \sum_{\substack{(w_s;p_s) \in P_s \\ p_s[x_{sk}] = 1}} \frac{w_r w_s g_{ji}^h k_i}{S[x_{qi}] S[x_{rj}] S[x_{sk}]}. \quad (8)$$

The fact that P_r and P_s satisfy property B ensure that

$$\sum_{(w_r; p_r) \in P_r} w_r = S[x_{rj}]$$

$$p_r[x_{rj}] = 1$$

and

$$\sum_{(w_s; p_s) \in P_s} w_s = S[x_{sk}]$$

$$p_s[x_{sk}] = 1$$

The fact that g_{ji} and h_{ki} are solutions to their respective transportation problems ensure that

$$\sum_{\substack{j \\ S[x_{rj}] > 0}} g_{ji} = \sum_{\substack{k \\ S[x_{sk}] > 0}} h_{ki} = S[x_{qi}].$$

Rearranging expression (8) and using these identities yields

$$\sum_{\substack{j \\ S[x_{rj}] > 0}} \frac{g_{ji}}{S[x_{rj}]} \sum_{\substack{(w_r; p_r) \in P_r \\ p_r[x_{rj}] = 1}} \frac{w_r}{S[x_{qi}]} = \sum_{\substack{k \\ S[x_{sk}] > 0}} \frac{h_{ki}}{S[x_{sk}]} \sum_{\substack{(w_s; p_s) \in P_s \\ p_s[x_{sk}] = 1}} w_s$$

$$= S[x_{qi}].$$

Therefore $T_q[x_{qi}] = S[x_{qi}]$ for all hosts i . Because every p_q in P_q has $p_q[x_{qi}] = 1$ for exactly one i , it follows that the sum of all weights in P_q is

$$\sum_{(w; p) \in P_q} w = \sum_i \sum_{\substack{(w; p) \in P_q \\ p[x_{qi}] = 1}} w = \sum_i T_q[x_{qi}] = \sum_i S[x_{qi}] = 1$$

Therefore, P_q has property A.

To show that $T_q[t_{rj}] = S[t_{rj}]$, it is necessary to first prove a lemma.

Lemma 1: If the vector g is the solution to the transportation problem between nodes r and q which was solved while constructing P_q , then $S[t_{rj}] = \sum_i (1 - \delta_{ij}) g_{ji}$ for any j .

Proof: Suppose $0 < S[x_{rj}] \leq S[x_{qj}]$ and $g_{ji} > 0$ for some i and j such that $i \neq j$. Construct g' , a new solution to the transportation problem, in the following way:

$$g'_{jj} = g_{jj} + g_{ji}$$

$$g'_{ji} = 0$$

$$g'_{kj} = g_{kj} - \frac{g_{ji} g_{kj}}{S[x_{qj}] - g_{jj}} \quad k=1, \dots, j-1, j+1, \dots, M$$

$$g'_{ki} = g_{ki} + \frac{g_{ji} g_{kj}}{S[x_{qj}] - g_{jj}} \quad k=1, \dots, j-1, j+1, \dots, M$$

$$g'_{ij} = g_{ij} \text{ for all other cases}$$

It is obvious that g'_{jj} , g'_{ji} , and g'_{ki} are non-negative. From the fact that $g_{ji} + g_{jj} \leq S[x_{qi}]$ it follows that g'_{kj} is non-negative, so all variables in g' are non-negative. From the fact that the solution g is feasible and the fact that

$$\sum_k g'_{kj} - \sum_k g_{kj} = g_{ji} - g_{ji} \sum_{k \neq j} \frac{g_{kj}}{S[x_{qj}] - g_{jj}} = g_{ji} - g_{ji} = 0$$

$$\Rightarrow \sum_k g'_{kj} = \sum_k g_{kj}$$

and (by similar reasoning)

$$\sum_k g'_{ki} = \sum_k g_{ki}$$

it follows that g' is feasible. The cost of g' will be lower than that of g by

$$g_{ji} + \frac{g_{ji}g_{ij}}{S[x_{qi}] - g_{jj}}$$

Hence, the original solution was not optimal. Therefore, in any optimal solution, if $S[x_{rj}] \leq S[x_{qj}]$, then $g_{jj} = S[x_{rj}]$ and $g_{ji} = 0$ for $i \neq j$. A similar argument will show that if $S[x_{rj}] \geq S[x_{qj}]$, then $g_{jj} = S[x_{qj}]$ and $g_{ij} = 0$ for $i \neq j$.

From the above argument, it follows that when $S[x_{rj}] \leq S[x_{qj}]$, then $g_{jj} = S[x_{rj}] = \sum_j g_{ji}$. This means that $\sum_i (1 - \delta_{ij}) g_{ji} = 0$. Similarly, if $S[x_{rj}] \geq S[x_{qj}]$, then $g_{jj} = S[x_{qj}]$, so $\sum_i (1 - \delta_{ij}) g_{ji} = \sum_i g_{ji} - g_{jj} = S[x_{rj}] - S[x_{qj}]$. Since $S[t_{rj}] \geq 0$ by constraint (7'), and since $S[t_{rj}] \geq S[x_{rj}] - S[x_{qj}]$ by constraint (6), it follows that $S[t_{rj}] \geq \sum_i (1 - \delta_{ij}) g_{ji}$. The assumption that V_r is positive ensures that $S[t_{rj}]$ will never be larger than is required by the constraints (6). Therefore, $S[t_{rj}] = \sum_i (1 - \delta_{ij}) g_{ji}$.

Q.E.D.

As before, $T_q[t_{rj}]$ is equal to the sum of the weights on all p_q 's which have $p_q[t_{rj}] = 1$. This sum is

$$\begin{aligned}
& \sum_{\substack{i \\ i \neq j \\ S[x_{qi}] > 0}} \sum_{\substack{(w_r; p_r) \in P_r \\ p_r[x_{rj}] = 1}} \sum_{\substack{k \\ S[x_{sk}] > 0}} \sum_{\substack{(w_s; p_s) \in P_s \\ p_s[x_{sk}] = 1}} \frac{w_r w_s g_{ji} h_{ki}}{S[x_{qi}] S[x_{rj}] S[x_{sk}]} \\
= & \sum_{\substack{i \\ S[x_{qi}] > 0}} \frac{(1 - \delta_{ij}) g_{ji}}{S[x_{rj}]} \sum_{\substack{(w_r; p_r) \in P_r \\ p_r[x_{rj}] = 1}} \frac{w_r}{S[x_{qi}]} \sum_{\substack{k \\ S[x_{sk}] > 0}} \frac{h_{ki}}{S[x_{sk}]} \sum_{\substack{(w_s; p_s) \in P_s \\ p_r[x_{rk}] = 1}} w_s
\end{aligned}$$

Using Lemma 1 and the same identities used to simplify expression (8), this reduces to

$$\sum_{\substack{i \\ S[x_{qi}] > 0}} (1 - \delta_{ij}) g_{ji}.$$

Since $S[x_{qi}] = 0$ implies $g_{ji} = 0$ (from the definition of the transportation problem), this is equal to

$$\sum_i (1 - \delta_{ij}) g_{ji} = S[t_{rj}].$$

Therefore, for all t_{rj} (and t_{sk} by similar argument), $T_q[t_{rj}] = S[t_{rj}]$ and $T_q[t_{sk}] = S[t_{sk}]$.

Now, it remains to be shown that $T_q[y] = S[y]$ for all variables y such that $p_r[y] = 1$ for some $(w_r; p_r) \in P_r$. (If $p_r[y] = 1$, there can be no p_s in P_s such that $p_s[y] = 1$.) $T_q[y]$ is obtained by summing the weights which correspond to solutions p in P_q which have $p[y] = 1$. This yields

$$\sum_{\substack{(w, p) \in P_q \\ p[y] = 1}} w$$

$$\begin{aligned}
&= \sum_i \sum_{S[x_{qi}] > 0} \sum_j \sum_{S[x_{rj}] > 0} \sum_{\substack{(w_r; p_r) \in P_r \\ p_r[x_{rj}] = 1 \\ p_r[y] = 1}} \sum_k \sum_{S[x_{sk}] > 0} \sum_{\substack{(w_s; p_s) \in P_s \\ p_s[x_{sk}] = 1}} \frac{w_r w_s g_{ji} h_{ki}}{S[x_{qi}] S[x_{rj}] S[x_{sk}]} \\
&= \sum_j \sum_{S[x_{rj}] > 0} \sum_{\substack{(w_r; p_r) \in P_r \\ p_r[x_{rj}] = 1 \\ p_r[y] = 1}} \frac{w_r}{S[x_{rj}]} \sum_i \sum_{S[x_{qi}] > 0} \frac{g_{ji}}{S[x_{qi}]} \sum_k \sum_{S[x_{sk}] > 0} \frac{h_{ki}}{S[x_{sk}]} \sum_{\substack{(w_s; p_s) \in P_s \\ p_s[x_{sk}] = 1}} w_s \\
&= \sum_j \sum_{S[x_{rj}] > 0} \sum_{\substack{(w_r; p_r) \in P_r \\ p_r[x_{rj}] = 1 \\ p_r[y] = 1}} w_r
\end{aligned}$$

Since each p_r in P_r will have $p_r[x_{rj}] = 1$ for exactly one j , the sum over j and the restriction $p_r[x_{rj}] = 1$ can be eliminated. The sum can then be reduced to

$$\sum_{\substack{(w_r; p_r) \in P_r \\ p_r[y] = 1}} w_r$$

This is exactly equal to $T_r[y]$, which by hypothesis is equal to $S[y]$, so $T_q[y] = S[y]$. A similar argument will show that $T_q[y] = S[y]$ for any y such that $p_s[y] = 1$ for some p_s in P_s . Therefore, P_q has properties B and C.

It has thus been shown that the set P_q has the properties A thru D as do the sets P_r and P_s . Since it is possible to construct such sets for the leaves of the query tree, it is possible to construct one for the root node, N , by induction. This set, P_N , will contain a group of integer, feasible

solutions. The non-integer solution S lies on the interior of the hyper-polyhedron defined by these vectors (i.e. is a convex combination of these vectors), so (by the argument on page 48) each of the integer solutions in P_N is an optimal solution.

An Algorithm for Finding Integer Solutions

The above argument proves the existence of an optimal, integer solution to the LP, but is entirely impractical as a method for finding such a solution. The following algorithm will quickly find an optimal, integer solution W , given an optimal, non-integer solution S .

1. Set the vector W to all zeroes.

Select an i such that $S[x_{Ni}] > 0$.

Set $W[x_{Ni}] = 1$.

2. Pick any operation r which has not been visited, but whose parent q has already been visited. (This is an arbitrary, top-down traversal of the tree.) If none such exists, stop. W is an optimal, integer solution.

3. Find the i such that $W[x_{qi}] = 1$. If $S[x_{ri}] \geq S[x_{qi}]$, let $j = i$. Otherwise, let j be any value such that $0 < S[x_{rj}] < S[x_{qj}]$. Set $W[x_{rj}] = 1$, and $W[t_{rj}] = (1 - \delta_{ij})$.

Go to step 2.

To show that this algorithm works it is sufficient to show that the solution it produces could be one of the solutions produced when constructing the set P_N . It is not necessarily one of those produced by a given execution of the construction

procedure because the solutions to the transportation problems are not necessarily unique.

Assume that, for some subtree with r at its root, w corresponds to some p_r in P_r . This means that $w[x_{ui}] = p_r[x_{ui}]$ for all i where u is either r or its descendant, and $w[t_{ui}] = p_r[t_{ui}]$ for all i where u is a descendant of r . Assume also that the same is true for P_s and the subtree with s at its root, and that r and s are children of q .

If $w[x_{rj}] = w[x_{qi}] = 1$, the conditions which were imposed on i and j in step 2 of the algorithm are sufficient to ensure that there exists a solution to the transportation problem between nodes r and q which has $g_{ji} > 0$. Similarly, if $w[x_{sk}] = 1$, it is possible to have $h_{ki} > 0$. From this and the derivation of P_q , it is clear that P_q could contain a solution of the form

$$p_r + p_s + \hat{x}_{qi} + (1 - \delta_{ij}) \epsilon_{rj} + (1 - \delta_{ik}) \epsilon_{sk}$$

This solution corresponds to w for the subtree with q at the root.

If q is a leaf and $w[x_{qi}] = 1$, then there must be a p in P_q such that $p = \hat{x}_{qi}$. It can therefore be shown by induction that w corresponds to some p in P_N for the subtree with root N . This means that w is the same as p , and since p is optimal, w is optimal.

VI. CONCLUSIONS

The described method for sampling a database to allow prediction of query performance can be fairly expensive, and will not be practical for small databases where the potential savings from optimization are small. For large databases, however, it will allow optimization based upon figures which are theoretically more valid than figures derived using an independence assumption. An important advantage of sampling is that quantitative estimates for the error in the estimates can be obtained.

A major part of the cost of sampling is the setup cost. A large amount of work must be done to build the samples and compute the weights which are used in interpreting the samples. Technically, these samples should be reconstructed whenever an update is made to the database. However, it is reasonable to expect that the statistical properties of the database should not change rapidly with time. Therefore, a sample, once constructed, can continue to be used even if the database has been updated until such time that it is observed to no longer reflect the status of the entire database.

REFERENCES

- Bernstein, P.A. "Synthesizing third normal form relations from functional dependencies," ACM Transactions on Database Systems, 1, 4 (December 1976), 277-298.
- Chu, W.W. "Optimal file allocation in a multiple computer system," IEEE Transactions on Computers, October 1969, 885-889.
- Codd, E.F. "A relational model of data for large shared data banks," Comm. ACM, 13, 6 (June 1970), 377-387.
- Codd, E.F. "Further normalization of the database relational model," in Data Base Systems, Courant Inst. Computer Science Symp. 6, R Rustin, Ed., Prentice-Hall, Englewood Cliffs, 1972, 33-64.
- Hammer, M. and Chan, A. "Index selection in a self adaptive data base management system," Proc. ACM-SIGMOD Conf. on Management of data, 1976.
- Levin, K.D. "Organizing distributed databases in computer networks," Tech Report No. 74-09-01 Dept of Decision Sciences, The Wharton School, University of Pennsylvania, (Ph.D. dissertation).
- Smith, J. and Chang, P. "Optimizing the performance of a relational algebra data base system," ACM-SIGMOD Workshop, San Francisco, CA (May 14, 1975).
- Vallarino, O. "On the use of bit maps for multiple key retrieval," Proc. ACM-SIGMOD Conf. on Data, Salt Lake City, March 1976.
- Wong, E. and Youssefi, K. "Decomposition - A strategy for query processing," ACM Trans. on Database Systems, 1, 3 (September 1976), 223-241.
- Wong, E. "Retrieving dispersed data from SDD-1: A System for Distributed Databases," Proc of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, 217-235.
- Yamane, T. Elementary Sampling Theory, Prentice-Hall, Englewood Cliffs, 1967.

APPENDIX A

Derivation of Formula For Probability of Correct Guess

Given that sampling has been used on the relations R and S to obtain p_R , d_R , p_S , and d_S , let

$$\mu_R = p_R |R| \qquad s_R = \frac{d_R |R|}{z}$$

$$\mu_S = p_S |S| \qquad s_S = \frac{d_S |S|}{z}$$

The actual volumes of $|(R|B_R)|$ and $|(S|B_S)|$ can then be thought of as normal distributions with means μ_R and μ_S and variances s_R^2 and s_S^2 . We say that $|(R|B_R)|$ is $N(\mu_R, s_R^2)$ and $|(S|B_S)|$ is $N(\mu_S, s_S^2)$. ($N(x, y)$ denotes a normal distribution with mean x and variance y .) Because the two samples were taken independently, it follows that $D = |(S|B_S)| - |(R|B_R)|$ is $N(\mu_S - \mu_R, s_S^2 + s_R^2)$, so

$$D' = \frac{D - (\mu_S - \mu_R)}{\sqrt{s_R^2 + s_S^2}} \text{ is } N(0, 1)$$

The probability of correct guess when $(S|B_S)$ is shipped is the probability that $|(R|B_R)|$ is greater than $|(S|B_S)|$. This is equal to

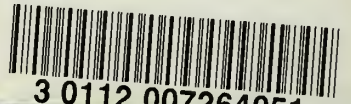
$$\begin{aligned} & P(|(R|B_R)| > |(S|B_S)|) \\ &= P(D < 0) \\ &= P\left(D' < \frac{\mu_R - \mu_S}{\sqrt{s_S^2 + s_R^2}}\right) \end{aligned}$$

$$= F\left(\frac{\mu_R - \mu_S}{\sqrt{s_R^2 + s_S^2}}\right)$$

$$= F\left(\frac{(p_R |R| - p_S |S|) z}{\sqrt{(d_S |S|)^2 + (d_R |R|)^2}}\right)$$

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER CAC Document Number 234	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) Optimization of a Relational-Algebra Query to a Distributed Database Using Statistical Sampling Methods		5. TYPE OF REPORT & PERIOD COVERED Thesis	
		6. PERFORMING ORG. REPORT NUMBER CAC #234	
7. AUTHOR(s) David A. Willcox		8. CONTRACT OR GRANT NUMBER(s) DCA100-75-C-0021	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Technical Support Activity 11440 Isaac Newton Square, North Reston, Virginia 22090		12. REPORT DATE August 1, 1977	
		13. NUMBER OF PAGES 69	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED	
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Copies may be obtained from the address in (9) above.			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) No restriction on distribution			
18. SUPPLEMENTARY NOTES None			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Distributed data management Relational database Sampling Query optimization			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A novel approach to the minimization of a relational-algebra query, where the relations of the database are distributed among several computers, is presented. A statistical sampling method is described which can be used to develop the parameters to an integer-linear programming (ILP) problem. An efficient method for solving the ILP is also presented.			

UNIVERSITY OF ILLINOIS-URBANA
510.84163C C001
CAC DOCUMENTS-URBANA
230-234 1977



3 0112 007264051